60/748,080

Unified Image Processing System

George Dalke, CTO
4G Color
Palo Alto, California
gdalke@4gcolor.com
www.4gcolor.com

Abstract
A comprehensive image processing system is derived from a common set of definitions and algorithms. This common approach results in consistency across all system functions such as authoring, selecting, combining and execution transforms. The formulation also allows each transform to be selectively applied and is suitable for chaining and combining a number of such transforms.

The unified approach leads to well-behaved, non-pathological transforms and gives consistency and predictability that greatly improves control and operation in complex distributed systems.

The solution consists of a fundamental set of definitions and algorithms from which a complete set of algorithms may be derived. By way of example, these are applied to several image processing system functions. It is asserted that these algorithms provide a complete unified fundamental approach to any image processing system.

Introduction

There are a number of requirements for a comprehensive image processing system. We group these under the following headings.

1. Transforms, The algorithms that transform the image.
2. Adaptation. The "run-time" selection, modification and combination of the transforms.
3. Authoring. Creating and editing transforms.
4. User interface. Selecting and adjusting transforms and intents.
5. Management. File management for transforms

While a number of algorithms are required to meet these requirements, they can all be derived from the following definitions and algorithms.

1. Every transform is parameterized and collected in a "set of transforms." That is, we develop a list of parameters for each transform in the set. We concatenate these lists for a composite parameter list (p-list). Such a p-list can execute any transform with any parameter. We will later state a number of functional requirements for these parametric transforms.
2. A set of selection functions is constructed. These functions, for example, express the "redness" or the "brightness" of image objects. We also apply a number of

ATTACHMENT 1

functional requirements on the set of selection functions. These requirements enable us to construct"selection logic" so that complex regions can be specified.
3. A p-list combination function, f, combines two p-lists.
4. A p-list scalar function, g. Combines a scalar with a p-list.

From these basic formulations, we derive the following operations and data structures

1. A list of p-lists called a v-array, V, where each p-list is associated with a selection function. The v-array contains sufficient information to selectively process an image.
2. A list of v-arrays called a q-array, Q. A q-array is the basis for combining multiple transforms into an equivalent executable transform.
3. A list of q-arrays called a w-array, W. A w-array is the basis for user interfaces and expert systems.
4. An algorithm F that combines arrays by repeatedly applying the p-list combination function to corresponding entries.
5. An algorithm G that combines scalars or scalar arrays to arrays by repeatedly applying the p-list scalar function g to appropriate entries.
6. A reduction algorithm K that reduces the dimension of an array by nested application of the p-list combination function f.

We will develop a notation for these concepts and formulate the algorithm for each requirement.

Underlying Concept

The underlying concept is that the ability to combine multiple transforms is equivalent to the ability to create and manage transforms. Suppose that we have a number of "basic" transforms, then control of a system is logically based on selecting arbitrary sets of these transforms. In order to create a generalized transform we created valid methods of generating sets of transforms.

Benefits of a unified approach

There are important benefits to Unified Image Transforms. Briefly, these are:

1. Efficient processing. Regardless of the number and complexity of a required transform sequence, it reduces to a single, well-behaved, transform.
2. Uniform performance throughout systems. The single transform can be optimized for software, firmware, and software.
3. The Unified Image Transform minimizes distortions and quantizing noise.
4. The ability to describe arbitrary sequences of selective transforms allows developing a corresponding syntax and vocabulary. This improves user interfaces, process control, and transform authoring.
5. Images, from creation to consumption, undergo a sequence of transforms. Yet, it is well known that each step introduces distortions and quantization noise[1]. The

[1] US Patent 5,809,178. "Elimination of visible quantizing artifacts in a digital image utilizing a critical noise/quantization factor." Anderson, Eric C; Dalke, George W.

more steps, the more the data deteriorates. But today's proliferation of digital imaging devices increases the number of steps and, consequently, the problems. Unified image transforms overcome this problem.

Historical Perspective

While imaging was a systems task, image-processing tasks were delegated to either specific devices or to general-purpose applications. Algorithms were generally buried in the hardware or device drivers and were constructed by the product design engineers. Images were processed and edited through a sequence of interactive operations by an expert using a workstation application.

Today, images are shared throughout a maze of devices and uses, each with different requirements. Device level optimization is not adequate for many of these combinations. Optimization is intra-device, not inter-device. Moreover, interactive editing and processing on a workstation is not suitable, either from a system viewpoint or a user viewpoint.

Image processing for device-to-device color calibration was relegated to so called "color management" systems. But these are operating system services and not suitable for distributed systems. Moreover, they are strictly limited to color matching and do not include other necessary and desirable image processes.

Typically, image processing is fragmented into independent processes that addressed several local parts of the image chain. They were optimized individually, with little regard to overall interaction. For example, a system that is color matched using ICC profiles is rendered invalid for even a simple operation such as changing the brightness of the image.

Requirements

The introduction organized the requirements under the following headings.

1. Transforms
2. Adaptation
3. Authoring
4. User interface

Transforms Requirements

1. **Selective**. Image transforms are selectively applied. The selection can be based on any image attribute. Regions of the image without the selected attribute are unchanged.
2. **Chainable**. A number of selective transforms, in an arbitrary sequence, can be applied with predictable results and minimum pathologies.
3. **Non-pathological**. None of the transforms or operations causes pathologies. The principal pathologies are: Quantization (posterization), gamut clipping, non-monotonic response, and color shift.

4. **Invertible**. All transforms and composed transforms must have a non-pathological inverse.
5. **Equivalent** execution in hardware, firmware or software is a requirement for distributed systems.

Adaptation (pre-execution)

Just prior to execution, the system may have collected a number of transforms to describe aspects of the execution. Typically, the system design will determine the number and purpose of each transforms. Some example cases are:

1. Device compensation transforms
2. Ambient condition compensation transforms
3. User preference transforms
4. Expert recommendations based on system conditions and data content
5. Transforms attached to the image by other devices or parts of the overall systems.

The adaptation a criterion is that these collected transforms must be combined to a single well behaved transform for processing the image.

Authoring

Authoring transforms is typically the job of a software engineer with adequate knowledge of image processing and color transforms. Most of the work is assuring the transform is well behaved over a reasonable range of operation. It is desirable to move transform authoring closer to the user. This is accomplished the following authoring tools and capabilities.

1. Transforms can be edited. That is, selected effects can be modified in a controlled continuous way.
2. Transforms can be combined to create new transforms
3. Transforms can be tested and evaluated

User Interfaces

Designing user interfaces is the job of a software engineer with knowledge of GUIs and of image processing and color transforms.

1. Uniform approach for any user interface
2. GUIs
3. Interactive controls
4. Declarative controls (language)

Management

1. Files of transforms
2. Retrieve
3. Share

Fundamental Definition and Algorithms

Perception criteria

There are a number of constraints that color perception imposes on image transforms. In deriving the Unified Image Transforms we only make use of the perception effect that discontinuous slopes in color space can cause the appearance of bands or contours. Therefore, we require that the Unified Image Transform is analytic in color space. That is, the transform and its derivatives are continuous. Additional perception criteria are used for developing specific transforms.

Image Notation

A bit mapped digital color image consists of an ordered arrangement of pixels. Typically, a pixel consists of rgb values given by the vector, $\mathbf{x} = (r, g, b)$. The space spanned by $\mathbf{x}$ is called the parameter space, X. (This is sometimes considered a color space but we carefully distinguish between parameter space and color space.)

An n pixel image, I, is given explicitly by the ordered sequence of N pixels,

$$I = (\mathbf{x}_1, ..., \mathbf{x}_i ..., \mathbf{x}_N).$$

Image Transform

An image transform t, maps an image, I, to a new image, $I' = t(I)$. In some cases an input pixel, $\mathbf{x}$, can be transformed directly to an output pixel, $\mathbf{x}' = t(\mathbf{x})$. Such an image transform is called a "point transform" since the new pixel depends only on the original pixel.

However, the new pixel can depend on other factors such as the values of neighborhood pixels. These factors can be collected in ordered sequences of metadata, **m**, and of neighborhood pixel values, **n**. Thus, a more general image transform is, $\mathbf{x}' = t(\mathbf{x}, \mathbf{m}, \mathbf{n})$. As a notational convenience **(m,n)** are omitted in this section but they are included in implementations if necessary.

An image, processed by such a transform, t, is given by

$$I' = \langle \mathbf{x}_1', \mathbf{x}_2', ..., \mathbf{x}_N' \rangle = \langle t(\mathbf{x}_1), t(\mathbf{x}_2), ..., t(\mathbf{x}_N) \rangle = t(I)$$

Multiple Transforms

Suppose the set of all available image transforms is, $T = \{t_1, t_2, \ldots . t_{nt}\}$. We can arrange these in a convenient order, say $T = (t_1, t_2, \ldots . t_{nt})$. A particular transform can be designated by it's index, k. Thus, an arbitrary, possibly redundant, sequence of np transforms can be specified by a sequence of indices, k_i as follows

$$k = (k_1, ..., k_{np}), \text{ where } k_i \in \{1, ..., nt\}$$

The transformed pixel, $\mathbf{x}' = k(\mathbf{x})$, is given by the sequence of transforms

$$\mathbf{x}_1 = t_{k1}(\mathbf{x}), \ \mathbf{x}_2 = t_{k2}(\mathbf{x}_1), \ \bullet\bullet\bullet, \ \mathbf{x}' = t_{knp}(\mathbf{x}_{np-1})$$

Or, equivalently, $k(\mathbf{x}) = t_{knp}(...t_{k2}(t_{k1}(\mathbf{x}))...)$.

As a notational convenience we use k as a sequence of indices and k(**x**) as the nested transforms corresponding to the sequence. Also note that if a transform, t_i, has neighborhood dependencies, each transform sequence must be applied for a region encompassing the neighborhood before applying the next transform[2].

One consequence of Unified Image Transforms is that certain neighborhood transforms (such as 2D convolution), can be solved as part of the General Transform and does not require sequential processing.

Selective Transforms

By selective transform[3], we mean that the transform is only applied to certain regions of an image, leaving the rest of the image unchanged. The selected region can be based on any image characteristic, including objects, color properties or image locations. Current art often uses so-called alpha planes to identify selected regions of an image.

For a particular characteristic, we define a corresponding discrete selector function, s(**x**), so that s(**x**) = 1 if the pixel **x** manifests the characteristic and s(**x**) = 0 if the pixel does not manifest the characteristic.

A selective transform t(x), relative to the selector, s(x) is given by:

$\mathbf{x}' = \mathbf{x}$, if $s(\mathbf{x}) = 0$

$\mathbf{x}' = t(\mathbf{x})$ if $s(\mathbf{x}) = 1$

Logical Functions of Selective Transforms

Suppose the set of all available selection functions is, $S = \{s_1, \ldots . s_{ns}\}$. We can arrange these in a convenient order, say $S = (s_1, \ldots . s_{ns})$. A particular selection function can be designated by it's index i.

Unlike transforms that are assembled as sequences, selectors are assembled as logic functions. Of course, after a transform is applied to a region defined by a logic function of selectors, then it becomes part of the transform sequence.

A logic function of selector functions can be constructed from a combination of logical operators, typically, intersection, union and negation. The construction of these logic functions is system dependent and may involve user input, work flow and image management considerations.

A logic function can be conveniently expressed as a sum of minterms. Suppose the selectors, $s_i(x)$ are analogous to minterms. That is, they are complete, mutually exclusive

---

[2] US Patent 6,028,611. "Modular digital image processing via an image processing chain." Anderson, Eric C; Chin, Gary; Dalke, George W.

[3] Us patent 366,027. "Method and means for color detection and modification." Dalke, George W; Puglsey, Peter C.

and the union is 1. Then any logic function, L(x) can be expressed as $L(\mathbf{x}) = \Sigma\, b_i z_i(\mathbf{x})$, where the coefficients, b_i, are either 0 or 1 and the summation is from 1 to ns.

For a logic function, L(x), we have a selective transform defined by:

$\mathbf{x}' = \mathbf{x}$, if $L(\mathbf{x}) = 0$
$\mathbf{x}' = t(x)$ if $L(\mathbf{x}) = 1$

For a chain of transforms described by the sequence, $k = (k_1, ..., k_{np})$ where the ki are indices of the ordered set of transforms, $T = (t_1, t_2, ... t_{nt})$, we have

$\mathbf{x}' = \mathbf{x}$, if $L(\mathbf{x}) = 0$
$\mathbf{x}' = k(x)$ if $L(\mathbf{x}) = 1$

The next step in this formulation is for the case consisting of a multiple chains of transforms each of which are applied to different logical sets of selectors. For this we need to develop additional notation, which is done later.

Analytic Selective Transforms

Unfortunately if $s(\mathbf{x})$ is a discrete function, the perception criteria may be violated. This problem is overcome by requiring $s(\mathbf{x})$ to be an analytic (continuously differentiable) function. It follows that the logic function is an analytic function and the meaning of logic functions for continuous functions must be defined.

For these functions to behave correctly, we need the following additional constraints

1. $0 \leq s(\mathbf{x}) \leq 1$, for all $\mathbf{x}$.
2. $\Sigma si(x) = 1$, summed over all i

We further require that for each selector function, si, there is at least one point, xi, such that si(xi) = 1. We call xi an inflection point.

These requirements lead directly to two consequences.

1. For an inflection point, xi, $s_i(x_j) = \delta_{ij}$, the kroniker delta function. That is, the inflection points are mutually exclusive.
2. slope(si(xj)) = 0. That is, all selectors have zero slopes at all inflection points. (Later, we relax this requirement for inflection points that are on a color gamut surface and only require the slope normal to the gamut surface to be 0.)

While the above requirements may appear quite restrictive they are not. A set of selection functions that do not meet the requirements can be modified as follows.

1. If a priori selectors, s_1 and s_2, are not mutually exclusive, then we can construct new selectors, $s_1 s_2$, $s_1(1-s_2)$, $(1-s_1)s_2$, and $(1-s_1)(1-s_2)$, that are mutually exclusively.
2. If the selector value at the inflection point is not 1, then we can create a new selector functions of $s_i(x)/s_i(x_i)$ that is equal to 1 at the inflection point.
3. If the$\Sigma s_i(x) \neq 1$, then add a new selector $s_n(x) = 1 - \Sigma s_i(x)$
4. If the derivatives of selectors are not zero at the inflection points, then we can sometimes find a global function, f, such that $f(s_i(x_i))$ does have zero derivatives at the inflection point.

Previously, when the selectors were considered discrete we stated that any combination of logic functions, L(x) can be expressed as $L(\mathbf{x}) = \Sigma\, b_i s_i(\mathbf{x})$, where the coefficients, b_i, are either 0 or 1 and the summation is from 0 to ns. Treating $s_i(x)$ as analytic functions with the properties given above gives the general form $L(\mathbf{x}) = \Sigma\, b_i s_i(\mathbf{x})$, where now the coefficients, b_i, are within the range (0,1).

Parametric Transforms

A parametric transform is defined as

$\mathbf{x}' = t(p,\mathbf{x})$ where p is a list of parameters called a p-list.

For a comprehensive system, we require that all possible transforms be included in a single parametric transform. Actually this is not difficult. Earlier we defined a sequence of transforms by a set of indices based on the ordered set of all transforms, $T = (t_1, t_2, \ldots . t_{nt})$. By including the index in the p-list, then any transform with any parameter can be specified.

Parametric Transform Constraints

So far, no limitations have been placed on the p-list, so any set of image processing algorithms can be expressed as a parametric transform. Now, we will add several constraints to the p-list. It may be impossible to meet these constraints for some algorithms. However, unless these constraints are met, then the algorithm is not appropriate for unified image processing systems.

The following constraints are placed on p-lists.

1. We require that there exists a p-list, p_0 that generates the identity transform, namely, $t(p_0,\mathbf{x}) = \mathbf{x}$.
2. We require that there exists a p-list, p_{inv} that generates the inverse transform, namely, $t(p_{inv}, t(p,\mathbf{x})) = \mathbf{x}$.
3. There exists a sequence of p-lists from p to p_0 and thence to p_{inv} so that the sequence of transforms t(p,x) transition smoothly.
4. We finally require that the p-list's are sorted into values that generated well behave, non-pathological transforms and those that don't. Those values that meet the requirements are the set of non-pathological p-values. Pathologies are defined in the next section

Color transform constraints

Generally, if the transform is analytic and invertible it is likely to be non-pathological unless the transforms operate on color images. Color image transforms have two additional constraints. These are color shift and gamut violations.

Rigorous derivation of these constraints requires mapping data values to a color space, so, here, the design rules that overcome these problems are stated as assertions, rather than derivations.

Gamut constraints

A gamut is a surface in a multi-dimensional space. We distinguish between the following cases.

1. If the gamut surfaces depend on a priori data ranges, they are called the "math gamuts."
2. If the envelope of data values determines the surfaces the transforms are called "data gamuts."
3. If the color range of some specified device determines the surfaces, they are called "device gamuts."
4. If the surfaces are determined by color perception limits, they are called "visual gamuts."

The design rule for gamut compliance is that the set of gamut values does not change although their coordinates may change. Essentially, this means that color values may not be moved onto, off of, or across a gamut, but, gamut colors can be rearranged and in-gamut colors can be rearranged.

Color shift constraint

A transform that avoids color shift (for a calibrated system) is expressed as

$\mathbf{x}' = a\mathbf{x} + b\mathbf{w}$, where

$\mathbf{x} = (r,g,b)$, an input data value

$\mathbf{x}' = (r',g',b')$, an output data value

$\mathbf{w} = (1,1,1)$ the visual white point

a,b are arbitrary scalars

For this system we construct an orthonormal basis **u**,**v**, and **h** relative to the plane a**x**+b**w**. It is convenient to take the first axis in the direction of **w**, namely, $\mathbf{u} = \mathbf{w}/|\mathbf{w}|$. The second axis is derived so that the uv plane is coplanar with **x**. That is, $\mathbf{v} = (\mathbf{x}-(\mathbf{x}\bullet\mathbf{u})\mathbf{u}) / |\mathbf{x}-(\mathbf{x}\bullet\mathbf{u})\mathbf{u}|$. We derive the third dimension, **h**, from the requirements that **h** is unitary and it is orthogonal to **x** and **w**. Thus, $\mathbf{h} = \mathbf{x}\otimes\mathbf{w}/|\mathbf{x}\otimes\mathbf{w}|$, where $\otimes$ is the vector cross product.

The component expansion of the **u**,**v** and **z** basis vectors for $\mathbf{x} = (r,g,b)$ is

$\mathbf{u} = (1,1,1)/\sqrt{3}$

$\mathbf{v} = (2r-g-b, 2g-r-b, 2b-r-g)/(6(r^2+g^2+b^2-rg-rb-gb))^{(1/2)}$

$\mathbf{h} = (g-b, b-r, r-g)/(2(r^2+g^2+b^2)-(rg+rb+gb))^{(1/2)}$

The coordinate transform of rgb to uvz is given by

$u = \mathbf{x}\bullet\mathbf{u} = (r+g+b)/\sqrt{3}$

$v = \mathbf{x}\bullet\mathbf{v} = (6(r^2+g^2+b^2) - 2(rg+rb+gb))^{(1/2)}$

$h = \mathbf{x}\bullet\mathbf{h} = 0$

The h=0 results is to be expected since, by design, **x** is in the uv plane. The uvh transform is not particularly useful as a computation space. Since h=0 for any **x**, the transform is degenerate and does not have a unique inverse. Moreover, v is somewhat costly to calculate.

However, the transform gives a simple and powerful method to constrain color shifts. Namely, a transform $\mathbf{x}' = t(\mathbf{x})$ does not shift color if $\mathbf{h}(\mathbf{x})\bullet\mathbf{x}' = 0$. The corresponding component relationship is

$$r'(g-b) + g'(b-r) + b'(r-g) = 0$$

Thus, a rgb transform can be constructed from an arbitrary transform of two color components and calculation of the third from this component relationship. Experience has shown that it is best to define transforms in terms of the largest and smallest pixel component and calculated the intermediate value from the component relationship.

Composition functions

The p-list composition function, f, is defined as

$$t(f(p_1,p_2),x) = t(p_2,T(p_1,x)),$$ where p1, p2 are p-lists

Based on the definitions of p0 and pinv we derive

$$f(p_0,p) = f(p,p_0) = p$$
$$f(p,p_{inv}) = p_0$$

In general, arbitrary parametric transforms do not compose to a transform of the same form. Thus, a composition function cannot be found. For example, if $t(p_1,x)$ and $t(p_2,x)$ are quadratic transforms, then $f(p_1,p_2)$ could be a forth order transform and, in such a case, $t(f(p_1,p_2),x)$ would not exist. A design challenge during implementation is to find transforms that compose. Fortunately such transforms can be found for most image processing tasks.

Scalar composition function

The scalar composition function, g(a,p) is defined as a smooth function where

$$g(1,p) = p$$
$$g(0,p) = p_0$$
$$g(-1,p) = p_{inv}$$

Based on this definition we note that

$$g(a,p0) = p0$$

Parameter Examples

Thus far we have been somewhat vague about the definition of parameters in parametric transforms. The only constraints that we have stated are that there exists parameter values that corresponds to the identity transform and the inverse transform. Some examples of p-list contents are:

1. normalized user parameters
2. device parameters
3. vectors
4. statistics
5. neighborhood values

Derived Algorithms

As stated above, algorithms for Unified Image Processing Systems can be derived from the basic definitions defined previously. First we will summarize the basic definitions and then derive additional algorithms

A summary of the basic 4G definitions and algorithms follows.

$x' = t(p,x)$, definition of t, a parametric transform,
where
x is an input pixel value, x = (r,g,b), (c,m,y,k), etc.
x' is an output pixel value, x' = (r',g',b'), (c',m',y',k'), etc.
p is a list of parameters (p-list) that defines all possible transforms

$t(p_0,x) = x$, definition of p_0, the identity p-list
$t(p_{inv}, t(p,x)) = x$, definition of p_{inv}, the inverse transform p-list

$t(f(p_1,p_2),x) = t(p_2,T(p_1,x))$, definition of f, a p-list/p-list composition function
note that,
$f(p_0,p) = f(p,p_0) = p$
$f(p,p_{inv}) = p_0$

g(a,p), definition of g, a scalar/p-list composition function,
where
$g(a,p_0) = p_0$
$g(0,p) = p_0$
$g(1,p) = p$
$g(-1,p) = p_{inv}$

$S(x) = (s_1(x), \ldots, s_n(x))$, a complete set of selector functions,
where
$\text{sum}(s_i(x)) = 1$
$0 \le s_i(x) \le 1$
there exists a value x_j such that $s_i(x_j) = \delta_{ij}$, the kroniker delta function
the slope of s_i at x_i is zero unless x_i is on a gamut.

Basic definitions for derived algorithms

Earlier we defined a basic structure for transforms, the p-list. We will need to create multi-dimensional arrays of p-lists. The naming conventions for these arrays are shown in the following table.

Typical Notation	degree of array	name	size	comment
V	1	v-array	n	n is size of selector list
Q	2	q-array	m*n	m is arbitrary
W	3	w-array	p*m*n	p is arbitrary

A **v-array**, $V = (p_1, \ldots, p_n)$ is a list of p-lists. Each p-list corresponds to a selector function. There must be the same number of entries as there are fundamental selector functions.

A **q-array**, $Q = (V_1, \ldots, V_k)$ is a list of v-arrays. The number of entries is problem specific.

A **w-array**, $W = (Q_1, \ldots, Q_m)$ is a list of q-arrays. The number of entries is problem specific, but each entry has the same number of p-arrays.

We associate scalars, a, and scalar arrays, A, with each of the above structures. The dimension of the scalar array will be problem dependent.

We also associate scalars, b, and scalar arrays, B, with the selector functions $S(x) = (s_1(x), \ldots, s_i(x), \ldots, s_n(x))$, Complex selection regions are logic functions of s_i and corresponding scalar arrays are calculated from the logic functions.

Basic derived algorithms

The **combination algorithm**, F, combines two similar structures by repeated application of the p-list composition function f to corresponding entries. Thus we have:

$p = f(p_1,p_2)$, combination of p-lists
$V = F(V_1,V_2)$, combination of v-arrays
$Q = F(Q_1,Q_2)$, combination of q-arrays
$W = F(W_1,W_2)$, combination of w-arrays

The **scalar combination algorithm**, G, combines a scalar or scalar array with one of the structures by repeated application of the scalar p-list combination function g(a,p). By convention, if the degree of the scalar array and the p-list array are the same, then the function, g, is applied to corresponding terms. If the degree of the scalar array is less than the degree of the p-list array, then repeated use of scalars is implied. Thus we have

$p = g(a,p_1)$, the scalar combination function
$V = G(a,V_1)$, the scalar combination algorithm for a scalar and a p-vector, where $V=(g(a,p_1), \ldots, g(a,p_n))$
$V = G(A,V1)$, the scalar combination algorithm for a scalar array (of degree 1) and a p-vector, where $V=(g(a_1,p_1), \ldots, g(a_n,p_n))$
$Q = G(A,Q)$, scalar combination of p-arrays
$W = G(A,W)$, scalar combination of q-arrays

The **inversion algorithm** is a special case of the scalar combination algorithm, based on the definition of g, we have

pinv = g(-1,p), by definition. And consequently
Vinv = G(-1,V)
Qinv = G(-1,Q)
Winv =G(-1,W)

The **reduction algorithm**, K, reduces the degree of the p-array by sequentially applying the combination algorithm. Thus we have

$p = K(V) = f(\ldots f(f(p_1,p_2), p_3), p_4),\ldots p_n)$, reduction of a v-array to a p-list
$V = K(Q) = F(\ldots F(F(V_1,V_2), V_3), V_4),\ldots V_n)$, reduction of a q-array to a v-array
$Q = K(W) = F(\ldots F(F(Q_1,Q_2), Q_3), Q_4),\ldots Q_n)$, reduction of a w-array to a q-array

Fundamental Algorithms for an Image Processing System

Execution
At execution time, the selected v-array, V, is used to process the image as follows

x' = t(K(G(S(x),V)), x) , where (as before)
x = an input pixel value, typically x = (r,g,b)
x' = an output pixel value
t is the parametric transform
V is a v-list consisting of n p-list structures
$S(x) = (s_1(x),\ldots,s_n(x))$ is a complete set of fundamental selectors
G is the scalar combination algorithm
K is the reduction algorithm that maps V into a p-list

Combining processes at execution time
Just prior to execution, the system may have collected a number of v-arrays to describe aspects of the execution. Typically, the system design will determine the number and purpose of each v-array. Some typical uses are:

1. Device compensation transforms
2. Ambient condition compensation transforms
3. User preference transforms
4. Expert recommendations based on system conditions and data content
5. Transforms attached to the image by other devices or parts of the overall systems.

For a system with k such v-arrays, the v-arrays are accumulated in a q-array, namely
$Q = (V_1, \ldots, V_k)$

Also in typical systems, a strategy of combining multiple transforms may have weights assigning the relative importance of each v-array. These weights also may be adapted by an expert system. In general, the weights consist of k scalars, given by:
$A = (a_1, \ldots, a_k)$

The algorithm for combining the q-array to a single v-array for execution is

$V = K(G(A,Q)$, where

G is the scalar combination algorithm

K is the reduction algorithm that maps Q into V

The combined execution transform thus becomes:

$x' = t(K(G(S(x), K(G(A,Q))), x)$

Device to device correction

We assert that system device correction can be calculated from forward and reverse chains of transforms. One problem is that by convention, device characterizations include pathologies. For this model we consider that pathological behavior of devices is embedded in device drivers, but the non-pathological portion of the device characteristics be given as a v-array and, thereby, can be combined with other devices.

Unified User Interface

User interfaces often face the problem that the best graphic doesn't correspond to well-behaved transforms and that well-behaved transforms don't have the best graphics. The reason is likely that optimizing transforms is a different design skill than optimizing GUIs. The unified approach, based on the common derivation consists of the following elements.

1. Uniform design approach for any user interface
   a. GUIs
   b. Interactive controls
   c. Declarative controls (language)
2. Consistent, predictable results
3. Standard range of operation

Uniform approach to user interfaces

The unified approach to user interfaces is to separate control capability from the specific control mechanisms. Thus, regardless of the type of interface, the following elements are specified. Moreover, several stages of different or redundant user interfaces can be combined with this approach.

1. A multi-dimensional set of user instructions
2. A logical specification of selectors
3. An assignment of selector relative importance
4. Developing a v-array for each instruction or control

Although a multi-dimensional transform is developed, the process is a sequence of single dimensional operations. The import and considerations are constraints, those things that are not to be changed, and selector weights, the relative effect on those thing that are to be changed.

The elements are descried mathematically as follows:

$U = (u_1, \ldots, u_k)$ is the multi-dimensional set of user instructions, where
$-1 \le u_i \le 1$,
$u_i = 0$ is the identity value

The scale of ui is so that the range (-1,1) is non-pathological. Generally, we want controls to operate over the largest possible non-pathological range. For this, we use the extreme non-pathological values.

Associated with each entry, ui, is the following

1. A logical set of selectors and selector weights. As derived earlier, an arbitrary logical set of selectors and selector weights can be represented by n parameters Bi $= (b_1, \ldots, b_n)$ where $0 \le b_i \le 1$. These logical regions can, in turn, be constructed with a variety of user interfaces and/or derived from image data statistics or features.
2. A set of v-arrays and weights giving the relative effect of each v-array for the overall transform. Specifically $Q_i = (V_1, \ldots, V_m)$ are m v-arrays along with $A_i = (a_1, \ldots, a_m)$ are m weights.
3. For each entry, we develop a resulting v-array given by $V_i = G(B_i, G(A_i, Q_i))$

Thus, for the user array U, we have developed a q-array, $Q = (V_1, \ldots, V_k)$. Prior to execution, we calculate a resulting v-array, $V = G(U,Q)$

(Note. This formulation applies selectors to v-arrays, which implicitly have already had a selection function applied. This is intentional. As an example, if a V-array's function is to brighten/darken red things, and the logical selector was set to only choose blue things, then the control, u, would have no effect. The point is that the system works perfectly, it was just asked to do an uninteresting job.)

Interactive GUIs

Interactive user interfaces provide visual feedback while controls are being adjusted. They feedback is sufficiently rapid so that the user can set the control. Typically, the feedback is the image or portion of the image so the results can be seen. Additionally it may consist of graphs and curves providing feedback information

Declarative

Declarative user interfaces emphasize predictability and control, rather than a visual feedback loop. Instead of adjusting a control until a desirable results is obtained. The required results are articulated. The basis for declarative control is the development of syntax, grammar and vocabulary. The power of declarative control is based, as was stated, on predictability non-pathologies.

Declarations can be spoken, written, developed with GUIs, or consequences of expert systems. The following is the general mapping of language to the parameters of a generalized user interface.

1. Nouns: selector logic functions or phrases
2. Adjectives; user parameters that modify selectors

3. Verbs: v-arrays
4. Adverbs: user parameters that modify v-arrays
5. Phrases: noun, verb, *optional* adjective, *optional* adverb